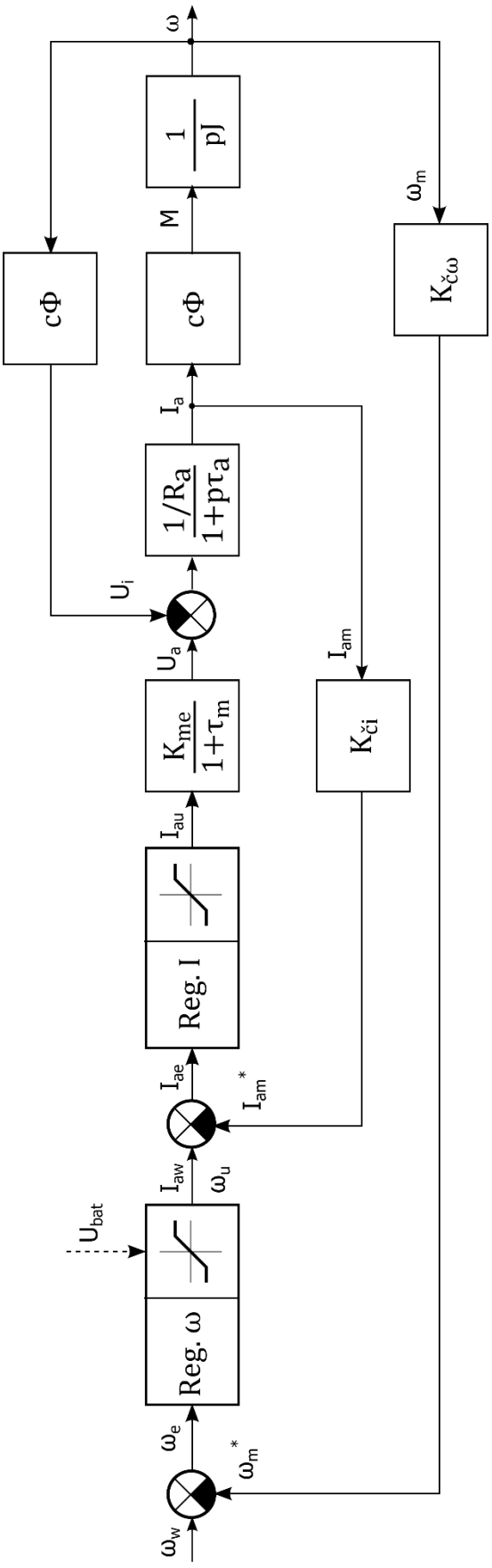
8 PŘÍLOHY

8.1 Přidělení funkcí pinů procesoru (pin assignment)

Pin číslo	Funkce	Návaznost
1	JTAG	TCK
2	RESET	JTAG – RESET
3	XTAL	Krystal
4	EXTAL	Krystal
5	GPIOC2	control_led1
6	TA0	poloha M1
7	TA1	poloha M1
8	ANA4	napětí baterie
9	ANA0	přijímač CH1
10	ANA1	přijímač CH2
11	ANA2	proud M1
12	ANA3	proud M2
13	GPIOC5	control_led2
14	GPIOB4/ANAB4	Universal IO pin 1
15	V_dda	3,3VA
16	V_ssa	GNDA
17	GPIOB0	sign I-M1
18	GPIOB1	sign I-M2
19	V_cap	extern capacitor
20	GPIOB2/ANAB2	Universal IO pin 2
21	GPIOB3/ANAB3	Universal IO pin 3
22	V_ss	GND
23	TA2	poloha M2
24	SCI_TXD0,SPI_SS*	Universal IO pin 5
25	SCI_RXD0,SPI_MISO	Universal IO pin 6
26	SPI_SCLK	Universal IO pin 7
27	SPI_MOSI/MISO	Universal IO pin 8
28	GPIOF0	control_led3
29	I2C_SCL1,SCI_TXD1,MSCAN	Universal IO pin 9
30	I2C_SDA1,SCI_RXD1,MSCAN	Universal IO pin 10
31	V_ss	3,3V
32	V_dd	GND
33	PWM_M1_2	motor 1 levý
34	PWM_M1_1	motor 1 levý
35	PWM_M2_2	motor 2 pravý
36	PWM_M2_1	motor 2 pravý
37	TA3	poloha M2
38	GPIOF1	M1_FAULT
39	GPIOE4	M2_FAULT
40	GPIOE5	M1_M2_D1
41	GPIOC14	Universal IO pin 11
42	GPIOC15	Universal IO pin 12
43	V_cap	externí kondenzátor
44	V_dd	3,3V
45	V_ss	GND
46	JTAG	TDO
47	JTAG	TMS
48	JTAG	TDI

8.2 Regulační schéma



8.3 Matlab skript na výpočet proudového regulátoru

```
clc;
clear;

%% vstupy
% paramtery motoru
Ra = 9.64;%Ohm
n0 = 5600;%/min
nn = 4100;%/min
La = 12.66e-3;%H
Mn_ = 50;%g*cm
Mk_ = 187;%g*cm
Ian = 350e-3;%A
Iak = 1.2;%A
Ia0 = 70e-3;%A


%ostatni paramtery
fpwm = 14.64e3;%Hz
g = 9.80665;%m/s2
U = 12;%V
Ukar = 2;%V

%cidlo proudu
Ki = tf((1)/(6.66/4))

%% mezivypocty
Mn = Mn_*0.00001*g;      %prevod jednotek g*cm na N*m
Mk = Mk_*0.00001*g;      %
cfi1 = (U-Ra*Ia0-Ukar)/(2*pi*(n0)/(60));   %urceni cfi ze stavu
naprazdno
cfi2 = (U-Ra*Ian-Ukar)/(2*pi*(nn)/(60));    %urceni cfi ze
stavu jmenoviteho
cfi3 = (Mn)/(Ian);                          %urceni cfi z
momentu jmen
cfi4 = (Mk)/(Iak);                          %urceni cfi z
momentu nakratko

cfi_ = [cfi1, cfi2, cfi3, cfi4];         %
cfi = mean (cfi_);                       %prumer cfi

Tme = 1/(2*fpwm);                %cas. konst. menice
TauA = La/Ra;                    %elmg. cas konst. kotvy

%% vypocty prenosu

%tf menice
Fme = tf([2*U], [Tme 1])
```

```
%tf kotvy
Fkot = tf([1/Ra], [TauA 1])

%urceni TauSigma
Tau = [Tme TauA];
TauSig = min(Tau)

%% vypocet regualtoru

%tf optimalni modul
Fo = tf(1, [2*TauSig*TauSig 2*TauSig 0])

%tf soustavy
Fs = Fkot * Fme * Ki

%tf regulatoru

Fr = Fo * (1/Fs)
Fr_ = minreal(Fr)
```

8.4 Program procesoru

```
#include "qs.h"
#include "occs.h"
#include "sys.h"
#include "adc.h"
#include "cop.h"
#include "crc.h"
#include "dac.h"
#include "gpio.h"
#include "hscmp.h"
#include "iic.h"
#include "intc.h"
#include "mscan.h"
#include "pwm.h"
#include "qtimer.h"
#include "vref.h"
#include "sci.h"
#include "spi.h"
#include "xbar.h"
#include "freemaster.h"

#include "gflib.h"                      //gen. fcs library (regulatory)

#define ABS(x) ((x)>0?(x):(-x))  //makro na absolutni hodntu cisla "x"


//voltage var
Frac16 ADCvoltage, ADCvoltage_flt;
Frac16 ADCvoltage_lim = FRAC16(0.17);
Frac16 ubatFilter_c0 = FRAC16(0.005), ubatFilter_c1;

//current var
Frac16 M1_I_flt, M2_I_flt;
Frac16 M1_I_offset = FRAC16(0.09), M2_I_offset = FRAC16(0.075);
Frac16 currFilter_c0 = FRAC16(0.3), currFilter_c1;
Frac16 I_limit = FRAC16(0.8);

static Frac16 M1_I, M2_I;
static unsigned short M1_I_unsgn, M2_I_unsgn;
static Frac16 M1_I_des = 0, M2_I_des = 0, M1_I_err, M2_I_err;
static GFLIB_CONTROLLER_PI_P_PARAMS_T M1_I_reg_params;
static GFLIB_CONTROLLER_PI_P_PARAMS_T M2_I_reg_params;
static Int16 M1_I_sat_flag = 0;
static Int16 M2_I_sat_flag = 0;

//comands var
Frac16 cmd1_in, cmd2_in, cmd1, cmd2, cmd1_const = FRAC16(0.68), cmd2_const =
FRAC16(0.645);
int cmd1_diff, cmd2_diff;
int cmd_enable = 1;
Frac16 cmd1_offset = FRAC16(0.766);
Frac16 cmd2_offset = FRAC16(0.8145);
Frac16 cmd_deadness = FRAC16(0.02);

Frac16 pwmFilter_c0 = FRAC16(0.01), pwmFilter_c1;

//speed var
unsigned int m1_spd_puls, m2_spd_puls;
unsigned int m1_quad_pulse, m2_quad_pulse, m1_quad_pulse_prev, m2_quad_pulse_prev;
int m1_quad_pulse_diff, m2_quad_pulse_diff;
Frac32 k_speed = 3000000;
Frac16 spdFilter_c0 = FRAC16(0.07), spdFilter_c1;
```

```
Frac16 spd1_correct = FRAC16(0.99), spd2_correct = FRAC16(0.98);

static Frac16 m1_speed, m2_speed, m1_speed_flt, m2_speed_flt;
static Frac16 M1_omg_des = 0, M2_omg_des = 0, M1_omg_err, M2_omg_err;
static GFLIB_CONTROLLER_PI_P_PARAMS_T M1_OMG_reg_params;
static GFLIB_CONTROLLER_PI_P_PARAMS_T M2_OMG_reg_params;
static Int16 M1_omg_sat_flag = 0;
static Int16 M2_omg_sat_flag = 0;

Frac16 omg_max = FRAC16(0.47);


//pwm var
Frac16 pwm0_23=0,pwm0_45=0;
Frac16 pwm1_23=0,pwm1_45=0;
short delay = 0;
Frac16 pwm0_dt=0, pwm1_dt=0, pwm0_dtz, pwm1_dtz, pwm0_dtz_flt, pwm1_dtz_flt;

//generator obdelniku
UWord16 gen = 0;
Frac16 gen_out = 0;
int reg_enable_0 = 0, reg_enable_1 = 0;
Frac16 gen_amp;
Frac16 test_out;
unsigned int gen_period = 5000;

//pozadi
Int32 a, b, c=20;
bool mot_block = 0;




void main (void)
{
    ioctl(SYS, SYS_INIT, NULL);
    ioctl(COP, COP_INIT, NULL);
    ioctl(GPIO, GPIO_INIT_ALL, NULL);
    ioctl(INTC, INTC_INIT, NULL);
    ioctl(QTIMER_B0, QT_INIT, NULL);
    ioctl(QTIMER_B2, QT_INIT, NULL);
    ioctl(QTIMER_B3, QT_INIT, NULL);
    ioctl(QTIMER_A0, QT_INIT, NULL);
    ioctl(QTIMER_A1, QT_INIT, NULL);
    ioctl(QTIMER_A2, QT_INIT, NULL);
    ioctl(QTIMER_A3, QT_INIT, NULL);
    ioctl(ADC, ADC_INIT, NULL);
    ioctl(EFPWM, EFPWM_INIT, NULL);    //inicializace pouzivanych periferii
    ArchIO.Adc.pwr &= ~3;              //power up ADC !!!
    ArchIO.Xbar.xbc3 = 0x0010;         //xbar - adc sync od pwm0
    //ArchIO.Xbar.xbc13 = 0x0603;      //xbar - vstupy TA2 a TA3 na TMR-B0 a TMR-B1
    ArchIO.Xbar.xbc14 = 0x0504;        //xbar - GPIOC9(pinhead7) na TMR-B2 a
    GPIOC10(pinhead8) na TMR-B3
    archEnableInt();
    FMSTR_Init();                       //inicializace debug softwaru freemaster

       //Reg.I1 init_params
        M1_I_reg_params.f16PropGain = FRAC16(0.05);
        M1_I_reg_params.f16IntegGain = FRAC16(0.01);
        M1_I_reg_params.i16PropGainShift = 3;
        M1_I_reg_params.i16IntegGainShift = 4;
        M1_I_reg_params.f32IntegPartK_1 = 0;
        M1_I_reg_params.f16UpperLimit = FRAC16(0.49);
       M1_I_reg_params.f16LowerLimit = FRAC16(-0.49);

       //Reg.OMG1 init_params
        M1_OMG_reg_params.f16PropGain = FRAC16(0.7);
```

```
        M1_OMG_reg_params.f16IntegGain = FRAC16(0.09);
        M1_OMG_reg_params.i16PropGainShift = 3;
        M1_OMG_reg_params.i16IntegGainShift = 0;
        M1_OMG_reg_params.f32IntegPartK_1 = 0;
        M1_OMG_reg_params.f16UpperLimit = I_limit;
       M1_OMG_reg_params.f16LowerLimit = -I_limit;

      //Reg.I2 init_params
        M2_I_reg_params.f16PropGain = FRAC16(0.05);
        M2_I_reg_params.f16IntegGain = FRAC16(0.01);
        M2_I_reg_params.i16PropGainShift = 3;
        M2_I_reg_params.i16IntegGainShift = 4;
        M2_I_reg_params.f32IntegPartK_1 = 0;
        M2_I_reg_params.f16UpperLimit = FRAC16(0.49);
       M2_I_reg_params.f16LowerLimit = FRAC16(-0.49);

      //Reg.OMG2 init_params
        M2_OMG_reg_params.f16PropGain = FRAC16(0.7);
        M2_OMG_reg_params.f16IntegGain = FRAC16(0.09);
        M2_OMG_reg_params.i16PropGainShift = 3;
        M2_OMG_reg_params.i16IntegGainShift = 0;
        M2_OMG_reg_params.f32IntegPartK_1 = 0;
        M2_OMG_reg_params.f16UpperLimit = I_limit;
        M2_OMG_reg_params.f16LowerLimit = -I_limit;

    while(1) //main loop
    {
        ioctl(COP, COP_CLEAR_COUNTER, NULL);

        //blokovanej mustek kdyz: HW-blok nebo napeti pilis male nebo SW-blok

        if(((ioctl(GPIO_B, GPIO_READ_DATA, NULL) & 0x0004 ) == 4) | (ADCvoltage_flt
<= ADCvoltage_lim))
        {
             ioctl(GPIO_E, GPIO_SET_PIN, BIT_5);           //blok obou mustku
             ioctl(GPIO_C, GPIO_SET_PIN, BIT_5);           //signalizace bloku
             mot_block = 1;
        }
        else
        {
             ioctl(GPIO_E, GPIO_CLEAR_PIN, BIT_5);         //deblok obou mustku
             ioctl(GPIO_C, GPIO_CLEAR_PIN, BIT_5);         //signalizace bloku
             mot_block =0;
        }

        FMSTR_Poll();

        cmd1_diff = (int)cmd1_in;
        cmd2_diff = (int)cmd2_in;

       a++;
       b++;
        if(a>=(c*ABS(cmd1_diff)))
        {
             ioctl(GPIO_C, GPIO_TOGGLE_PIN, BIT_2); //signalizace centrovani cmd1
             a=0;
        }

        if(b>=(c*ABS(cmd2_diff)))
        {
             ioctl(GPIO_F, GPIO_TOGGLE_PIN, BIT_0); //signalizace centrovani cmd2
             b=0;
        }

        M1_OMG_reg_params.f16UpperLimit = I_limit;
       M1_OMG_reg_params.f16LowerLimit = -I_limit;
```

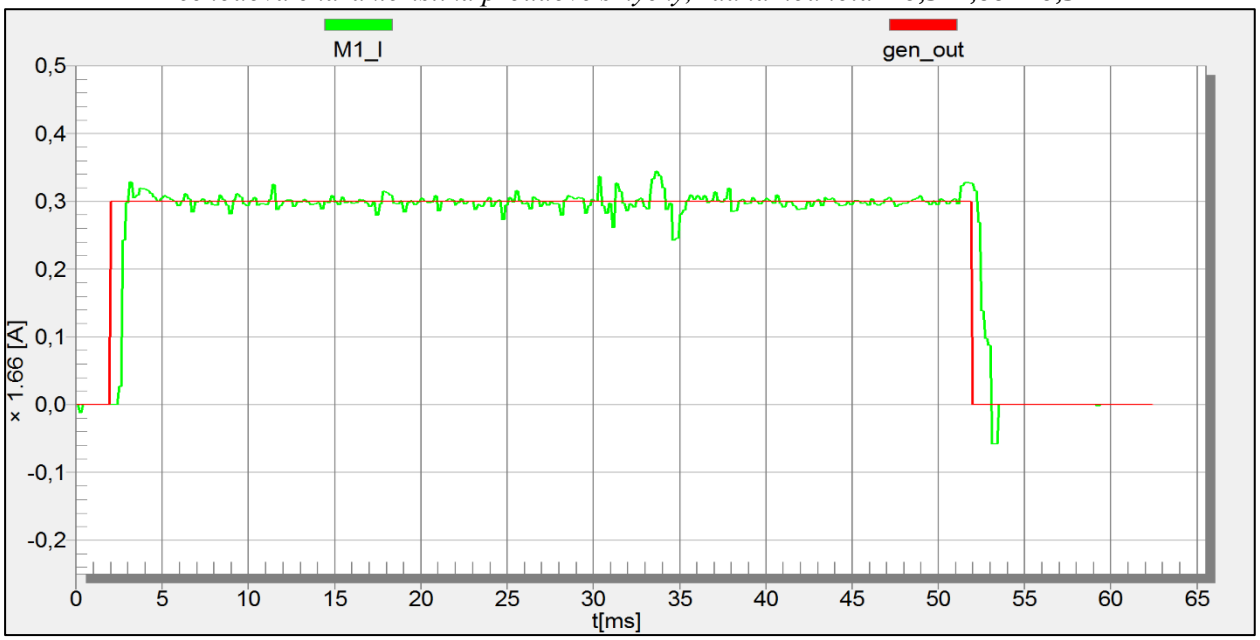
```
        M2_OMG_reg_params.f16UpperLimit = I_limit;
      M2_OMG_reg_params.f16LowerLimit = -I_limit;

      //filtr. const
      currFilter_c1 = FRAC16(1.0) - currFilter_c0;
      ubatFilter_c1 = FRAC16(1.0) - ubatFilter_c0;
      spdFilter_c1 = FRAC16(1.0) - spdFilter_c0;
    }
}

void pwm_isr (void)  //0,5kHz frekvence
#pragma interrupt saveall
{
      //U_batt - snimani a filtrace
      ADCvoltage = (Frac16)ioctl(ADC, ADC_READ_SAMPLE, 2);
      ADCvoltage_flt =
mac_r(L_mult(ubatFilter_c1,ADCvoltage_flt),ubatFilter_c0,ADCvoltage);

      //Otacky - zpracovani kvad. modu, vypocet rychlosti a filtrace
      m1_quad_pulse = ((unsigned int)ioctl(QTIMER_A1, QT_READ_COUNTER_REG, NULL));
      m1_quad_pulse_diff = (int)(m1_quad_pulse_prev - m1_quad_pulse);
      m1_quad_pulse_prev = m1_quad_pulse;
      m1_speed = (div_ls(k_speed,(Frac16)m1_spd_puls)*(m1_quad_pulse_diff > 0 ? 1:-
1));
      if(m1_quad_pulse_diff == 0) m1_speed = 0;

      m2_quad_pulse = ((unsigned int)ioctl(QTIMER_A3, QT_READ_COUNTER_REG, NULL));
      m2_quad_pulse_diff = (int)(m2_quad_pulse_prev - m2_quad_pulse);
      m2_quad_pulse_prev = m2_quad_pulse;
      m2_speed = (div_ls(k_speed,(Frac16)m2_spd_puls)*(m2_quad_pulse_diff > 0 ? 1:-
1));
      if(m2_quad_pulse_diff == 0) m2_speed = 0;

//    if(m1_speed > 0)
//          m1_speed = mult_r(m1_speed,spd1_correct);


      //cmd - pasmo necitlivosti
      if(cmd1_in >= cmd_deadness)
            cmd1 = cmd1_in - cmd_deadness;
      else if(cmd1_in <= -cmd_deadness)
            cmd1 = cmd1_in + cmd_deadness;
      else
            cmd1 = FRAC16(0.0);

      if(cmd2_in >= cmd_deadness)
            cmd2 = cmd2_in - cmd_deadness;
      else if(cmd2_in <= -cmd_deadness)
            cmd2 = cmd2_in + cmd_deadness;
      else
            cmd2 = FRAC16(0.0);

      //cmd saturace
      if(cmd1 >= omg_max)
            cmd1 = omg_max;
      else if (cmd1 <= -omg_max)
            cmd1 = -omg_max;

      if(cmd2 >= omg_max)
            cmd2 = omg_max;
      else if (cmd2 <= -omg_max)
            cmd2 = -omg_max;

      if(cmd_enable == 1)
      {
            M1_omg_des = cmd2 - cmd1;
            M2_omg_des = cmd2 + cmd1;
```

```
		//saturace zadane hodnoty otacek
		if(M1_omg_des >= omg_max)
			M1_omg_des = omg_max;
		else if(M1_omg_des <= -omg_max)
			M1_omg_des = -omg_max;
		if(M2_omg_des >= omg_max)
			M2_omg_des = omg_max;
		else if(M2_omg_des <= -omg_max)
			M2_omg_des = -omg_max;
	}
	else
	{
		M1_omg_des = M1_omg_des;
		M2_omg_des = M2_omg_des;
	}

	//omg. reg - regul. odchylka otacek
	M1_omg_err = M1_omg_des - m1_speed_flt;
	M2_omg_err = M2_omg_des - m2_speed_flt;

	//omg regulace
	if (mot_block == 0)
	{
		M2_I_des = GFLIB_ControllerPIp(M2_omg_err, &M2_OMG_reg_params,
&M2_omg_sat_flag);
		M1_I_des = GFLIB_ControllerPIp(M1_omg_err, &M1_OMG_reg_params,
&M1_omg_sat_flag);
	}
	ioctl(EFPWM_SUB3, EFPWMS_CLEAR_SUBMODULE_FLAGS, EFPWM_COMPARE_VAL0);

}


void adcEOSisr(void)	//frekvence fpwm - 14.64844 kHz
{
	#pragma interrupt saveall
	ioctl(ADC, ADC_CLEAR_STATUS_EOSI, NULL);

	//Proudy
	M1_I_unsgn = (ioctl(ADC, ADC_READ_SAMPLE, 0));	//cteni proudu M1
	M2_I_unsgn = (ioctl(ADC, ADC_READ_SAMPLE, 1));	//cteni proudu M2
	M1_I_unsgn = M1_I_unsgn << 2;			//bitovy shift (zesileni 4krat)
	M2_I_unsgn = M2_I_unsgn << 2;			//bitovy shift (zesileni 4krat)
	if(ioctl( GPIO_B, GPIO_READ_DATA, NULL) & 0x0001) M1_I = (Frac16)M1_I_unsgn *
(-1);  //prideleni smeru proudu
	else M1_I = (Frac16)M1_I_unsgn;
	if(!(ioctl( GPIO_B, GPIO_READ_DATA, NULL) & 0x0002)) M2_I =
(Frac16)M2_I_unsgn * (-1);		//prideleni smeru proudu
	else M2_I = (Frac16)M2_I_unsgn;


	m1_speed_flt =
mac_r(L_mult(spdFilter_c1,m1_speed_flt),spdFilter_c0,m1_speed);   //filtr otacek
	m2_speed_flt =
mac_r(L_mult(spdFilter_c1,m2_speed_flt),spdFilter_c0,m2_speed);   //filtr otacek

	M1_I_err = M1_I_des - M1_I;			//
	M2_I_err = M2_I_des - M2_I;			//vypocet reg. odchylky

	//pwm
	if (mot_block == 0)
	{
		pwm0_dt = GFLIB_ControllerPIp(M1_I_err, &M1_I_reg_params,
&M1_I_sat_flag);
		pwm1_dt = GFLIB_ControllerPIp(M2_I_err, &M2_I_reg_params,
&M2_I_sat_flag);
	}
```

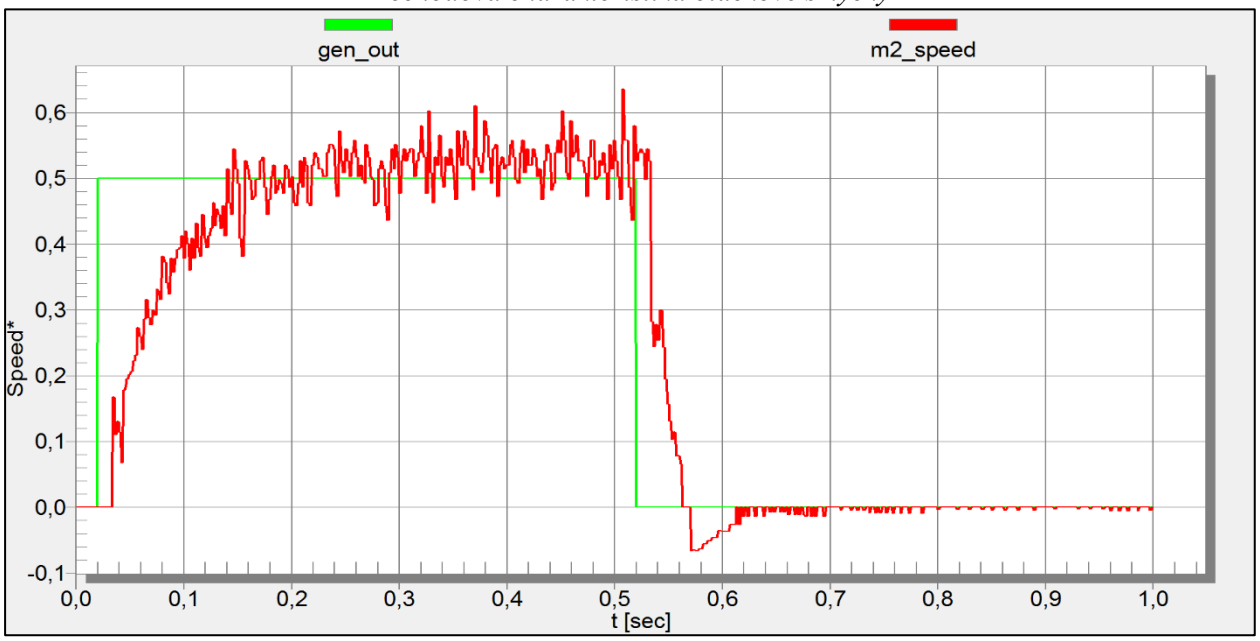
```
	pwm0_23 = FRAC16(0.5) - pwm0_dt;	//
	pwm0_45 = FRAC16(0.5) + pwm0_dt;	//

	pwm1_23 = FRAC16(0.5) + pwm1_dt;	//
	pwm1_45 = FRAC16(0.5) - pwm1_dt;	//vypocet hodnot pro VAL registry

	ioctl(EFPWM_SUB0, EFPWMS_CENTER_ALIGN_UPDATE_CHANNEL_23_FRAC, pwm0_23);
	ioctl(EFPWM_SUB0, EFPWMS_CENTER_ALIGN_UPDATE_CHANNEL_45_FRAC, pwm0_45);
	ioctl(EFPWM_SUB1, EFPWMS_CENTER_ALIGN_UPDATE_CHANNEL_23_FRAC, pwm1_23);
	ioctl(EFPWM_SUB1, EFPWMS_CENTER_ALIGN_UPDATE_CHANNEL_45_FRAC, pwm1_45);

	//update VAL regs
	ioctl(EFPWM_SUB0, EFPWMS_WRITE_VALUE_REG_0, delay);  //zapis spozdeni A/D
prevodu proudu

	ioctl(EFPWM, EFPWM_SET_LOAD_OK, EFPWM_SUBMODULE_0);	//
	ioctl(EFPWM, EFPWM_SET_LOAD_OK, EFPWM_SUBMODULE_1);	//potvrzeni zapisu
VAL registru PWM
}

void cmd1captureIsr(void) //f=50Hz		//snimani delky pulzu od cmd1 (smer)
#pragma interrupt saveall
{
	cmd1_in = (((Frac16)ioctl(QTIMER_B2, QT_READ_CAPTURE_REG, NULL))-
cmd1_offset);
	cmd1_in = mult_r(cmd1_in, cmd1_const);
	ioctl(QTIMER_B2, QT_WRITE_COUNTER_REG, 0x0000);
	ioctl(QTIMER_B2, QT_CLEAR_FLAG, QT_INPUT_EDGE_FLAG);
}


void cmd2captureIsr(void) //f=50Hz	//snimani delky pulzu od cmd2 (rychlost)
#pragma interrupt saveall
{
	cmd2_in = (((Frac16)ioctl(QTIMER_B3, QT_READ_CAPTURE_REG, NULL))-
cmd2_offset);
	cmd2_in = mult_r(cmd2_in, cmd2_const);
	ioctl(QTIMER_B3, QT_WRITE_COUNTER_REG, 0x0000);
	ioctl(QTIMER_B3, QT_CLEAR_FLAG, QT_INPUT_EDGE_FLAG);
}

void tmrA0_spd_isr(void)				//mereni otacek M1
#pragma interrupt saveall
{
	m1_spd_puls = (ioctl(QTIMER_A0, QT_READ_CAPTURE_REG, NULL));
	ioctl(QTIMER_A0, QT_WRITE_COUNTER_REG, 0x0000);
	ioctl(QTIMER_A0,QT_CLEAR_FLAG, QT_INPUT_EDGE_FLAG);
}

void tmrA2_spd_isr(void)				//mereni otacek M2
#pragma interrupt saveall
{
	m2_spd_puls = (ioctl(QTIMER_A2, QT_READ_CAPTURE_REG, NULL));
	ioctl(QTIMER_A2, QT_WRITE_COUNTER_REG, 0x0000);
	ioctl(QTIMER_A2,QT_CLEAR_FLAG, QT_INPUT_EDGE_FLAG);
}
```

8.5 Přechodové charakteristiky regulátorů

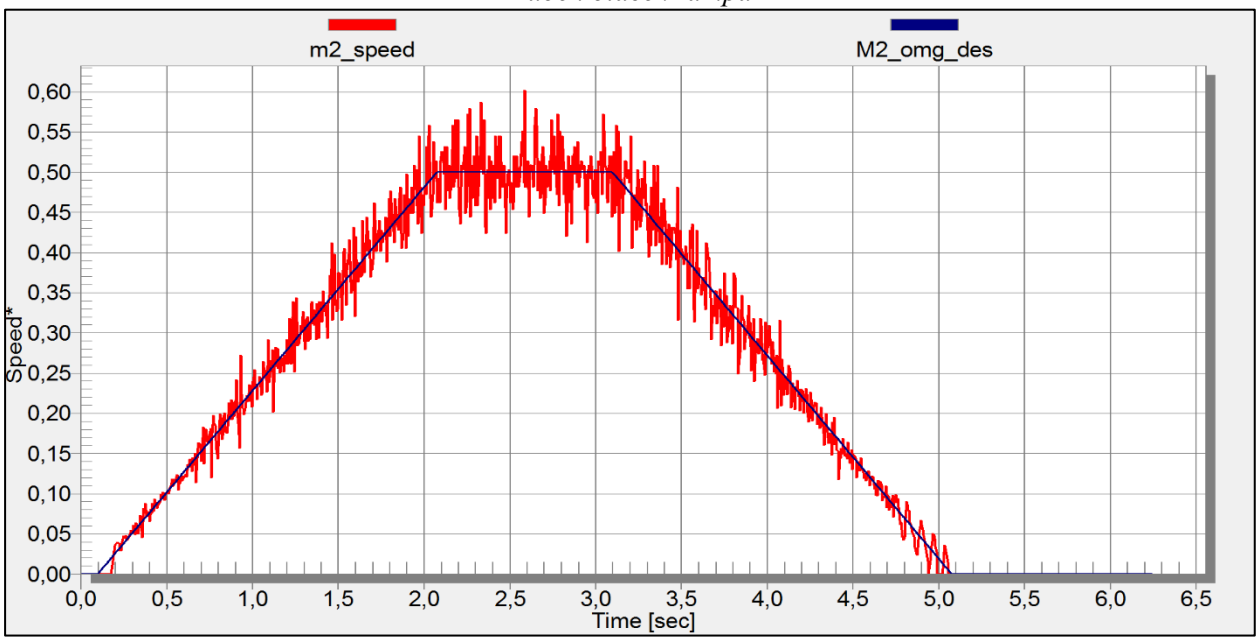
*Přechodová charakteristika proudové smyčky, žádná hodnota – 0,3*1,66 = 0,5 A*



Přechodová charakteristika otáčkové smyčky

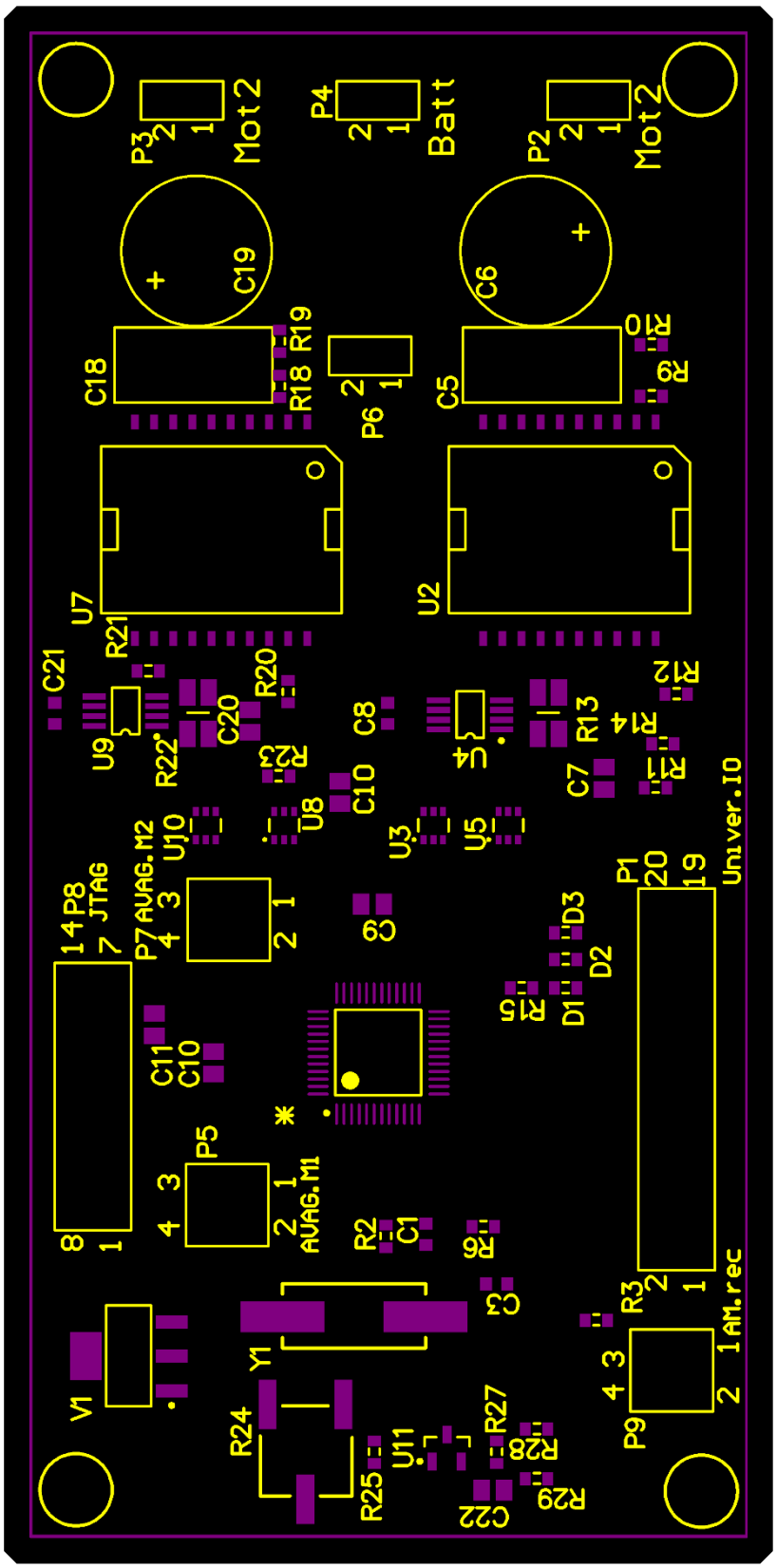


Průběh otáček rampa

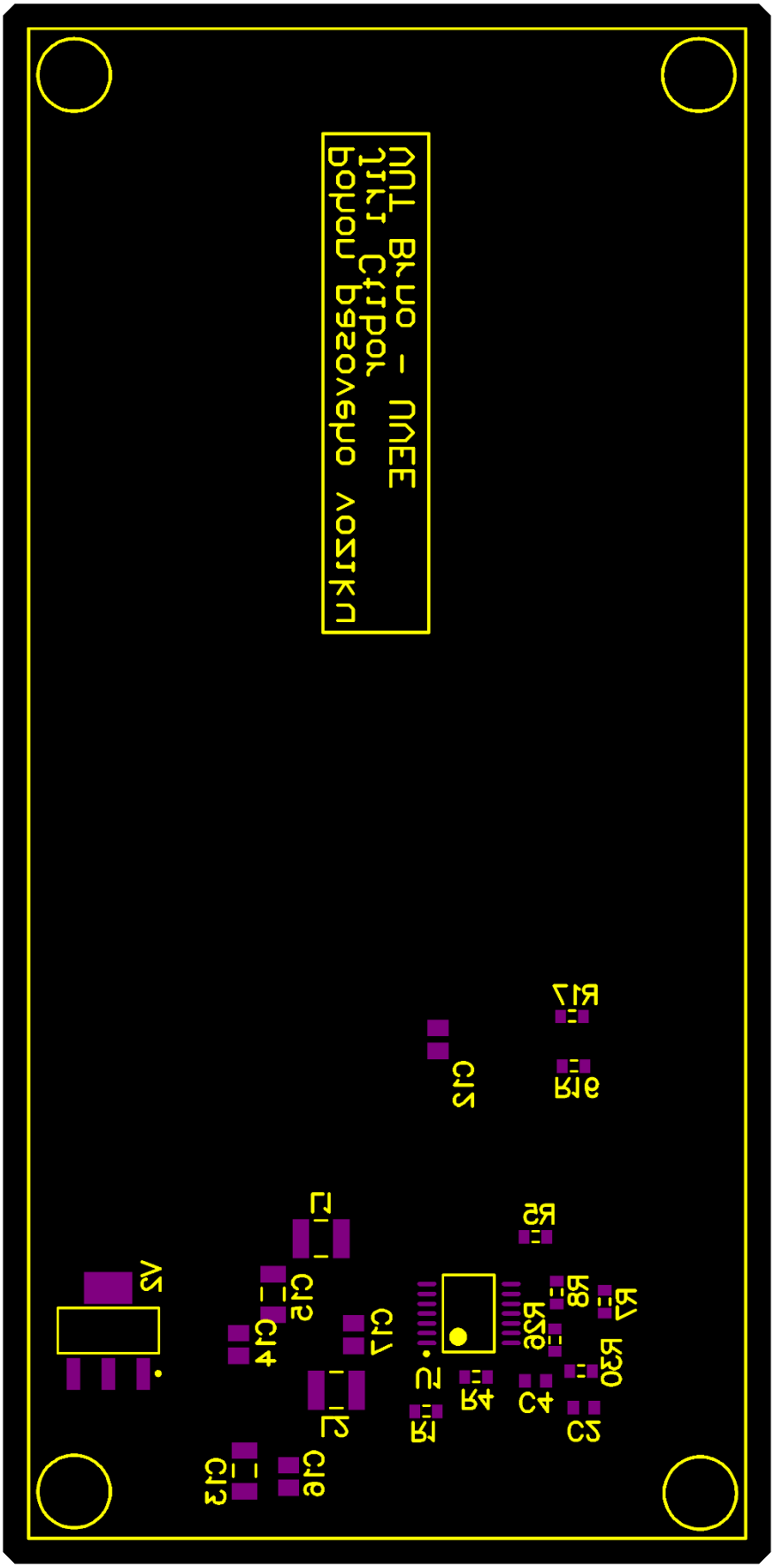


8.6 Gerber soubory DPS a schémata zapojení

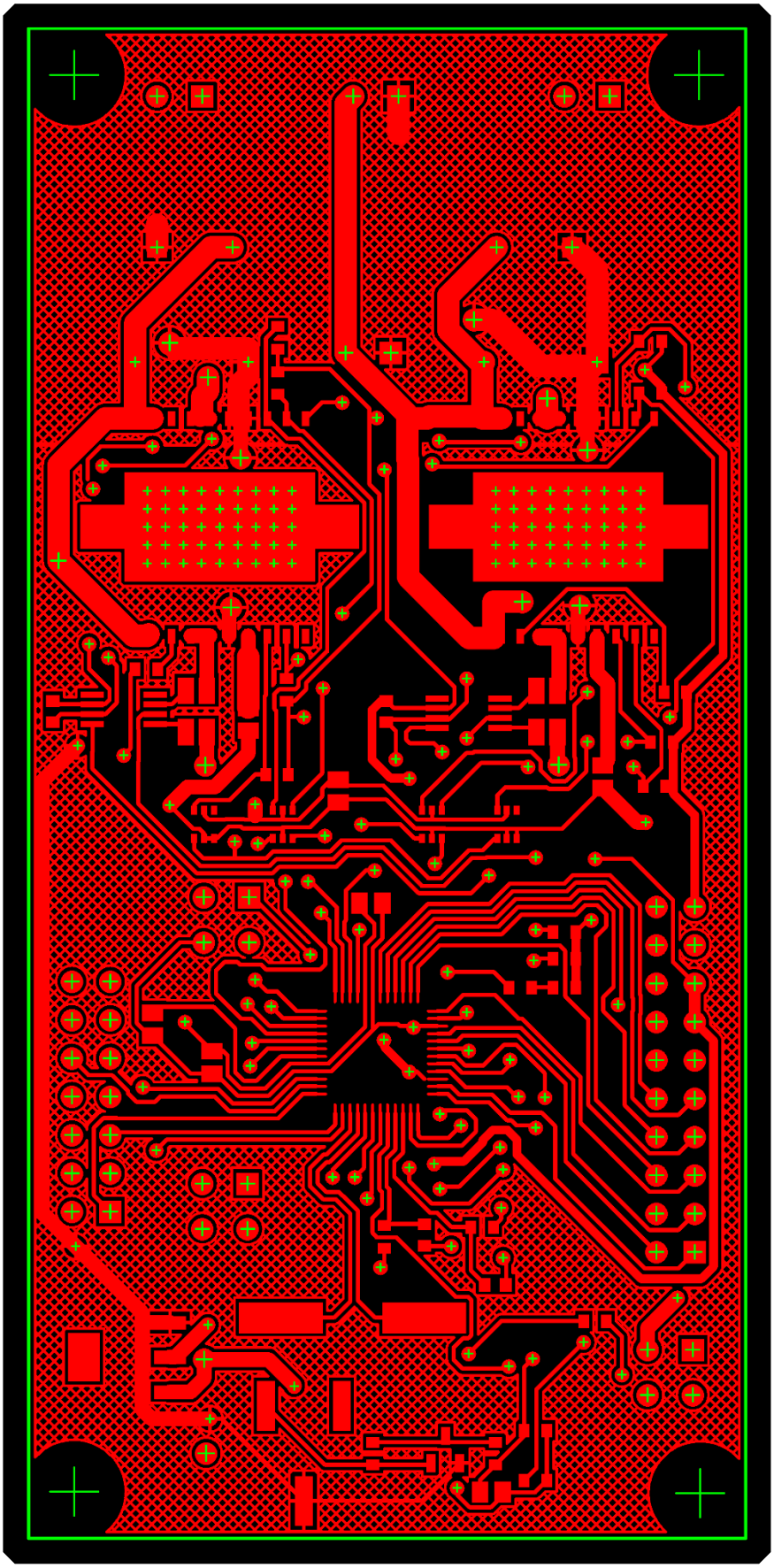
Rozmístění komponent vršek



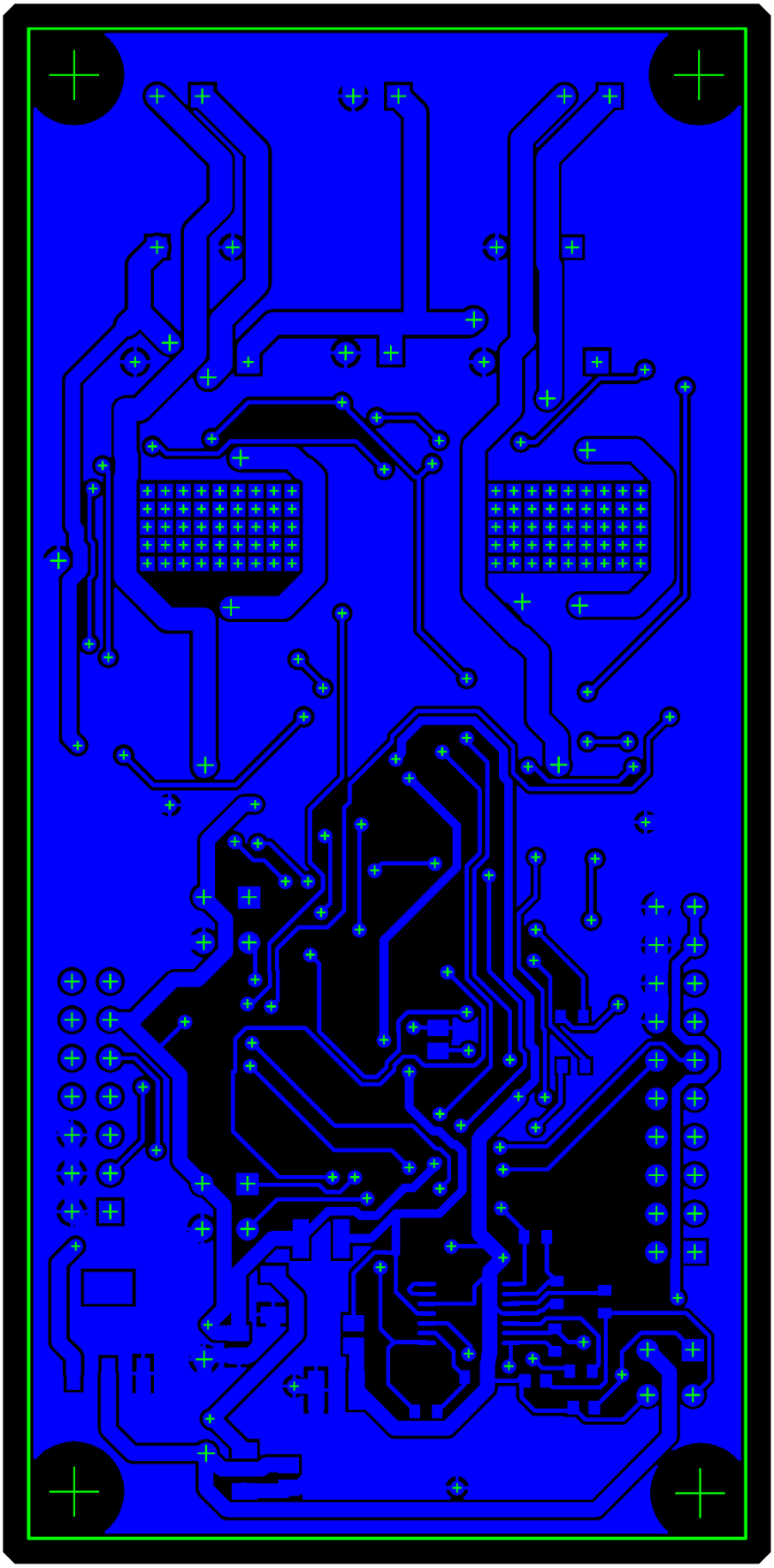
Rozmístění komponent spodek

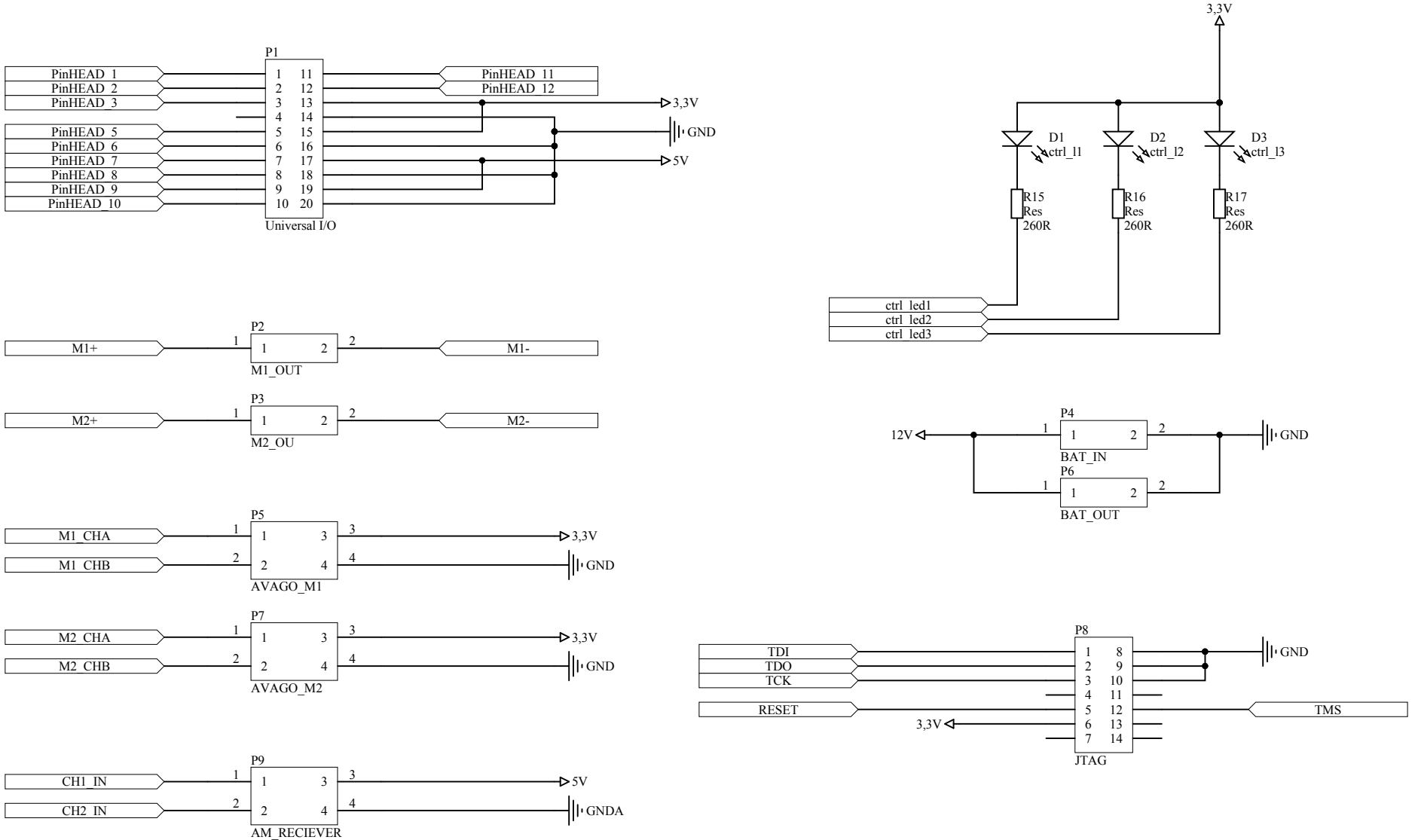


Spoje vršek

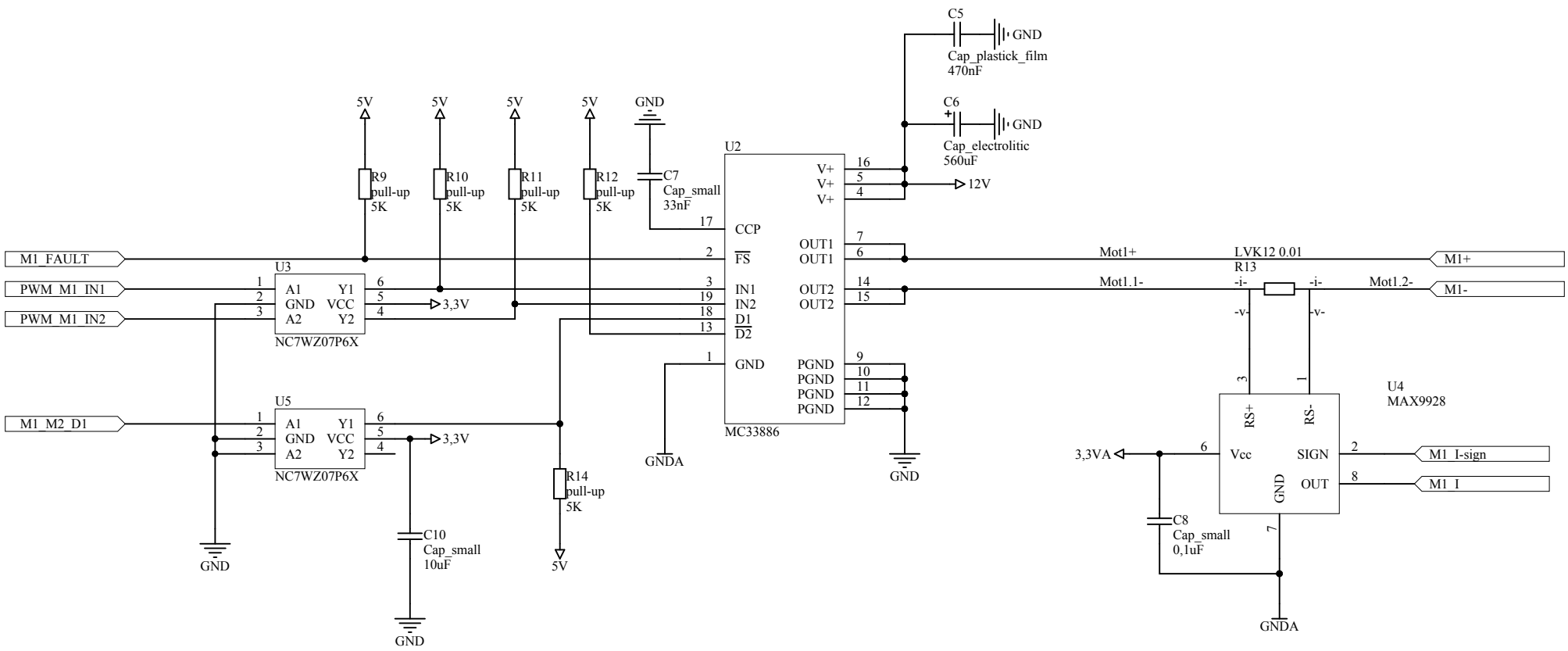


Spoje spodek

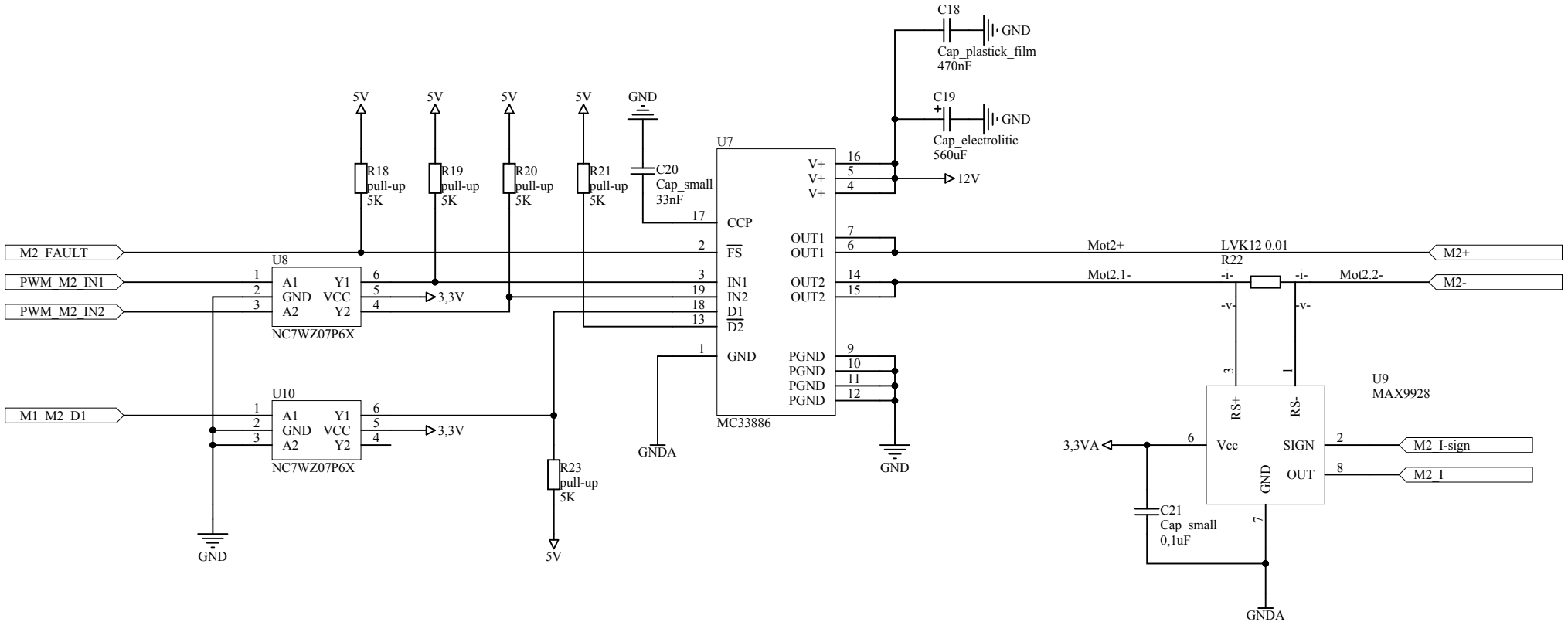




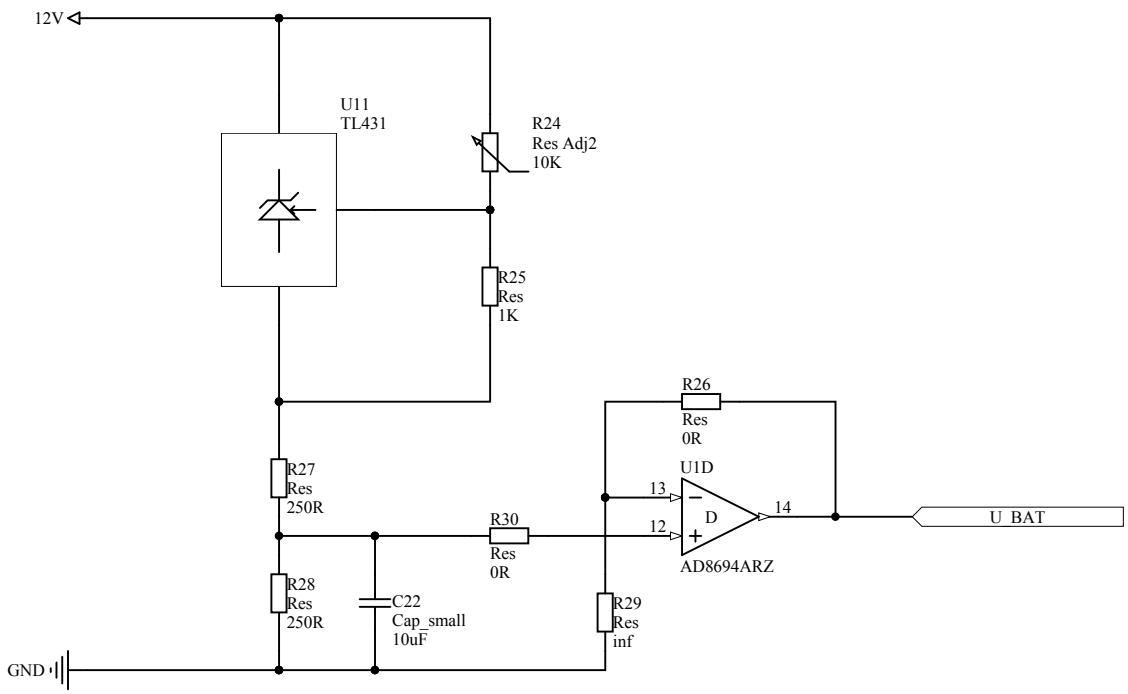
Title	**Input/Outputs**		
Size A4	Number **1.0**	Revision **1**	
Date: 28.5.2014		Sheet of **4/7**	
File: D:\Dokumenty\..\I_O.SchDoc		Drawn By: **Jiri Ctibor**	



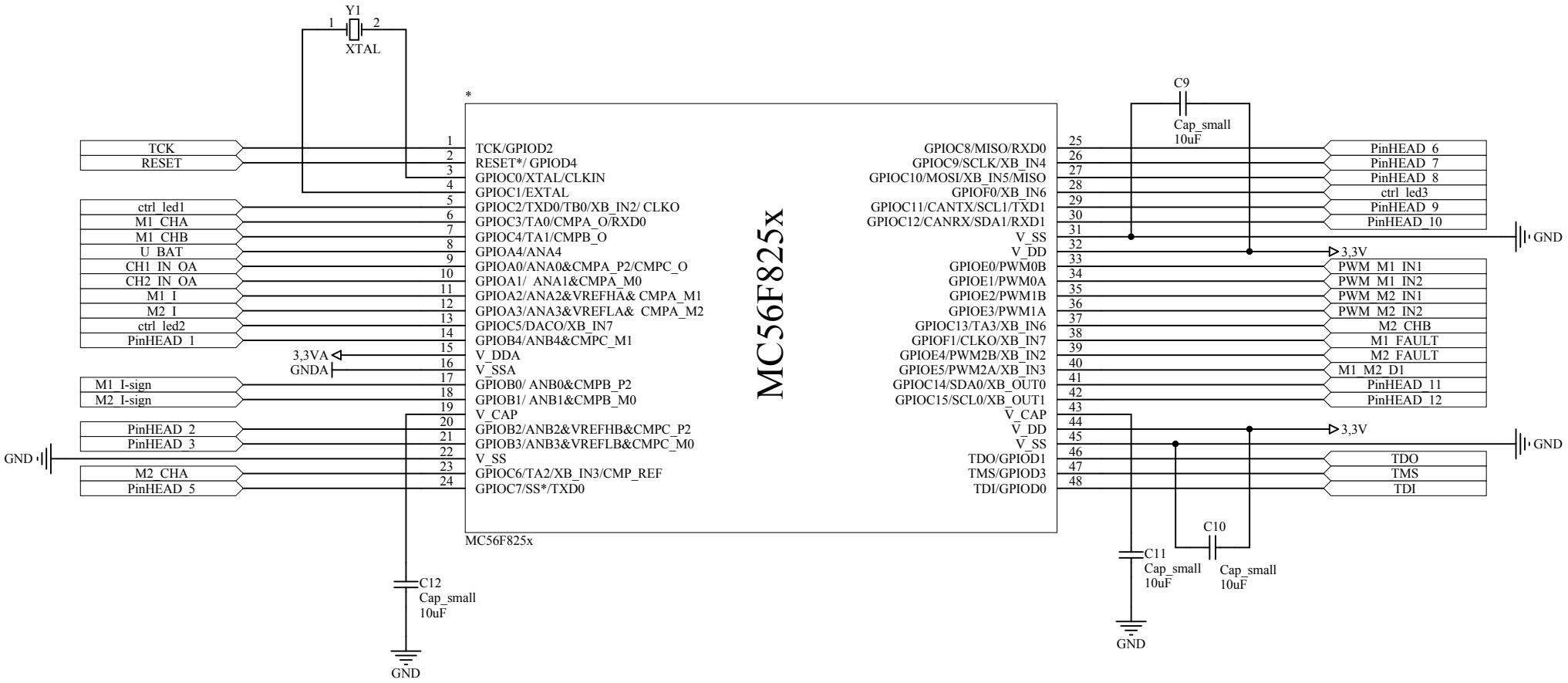
Title	**H-bridge M1**		
Size A4	Number **1.0**		Revision **1**
Date: 28.5.2014		Sheet of **2/7**	
File: D:\Dokumenty\..\menic_M1.SchDoc		Drawn By: **Jiri Ctibor**	



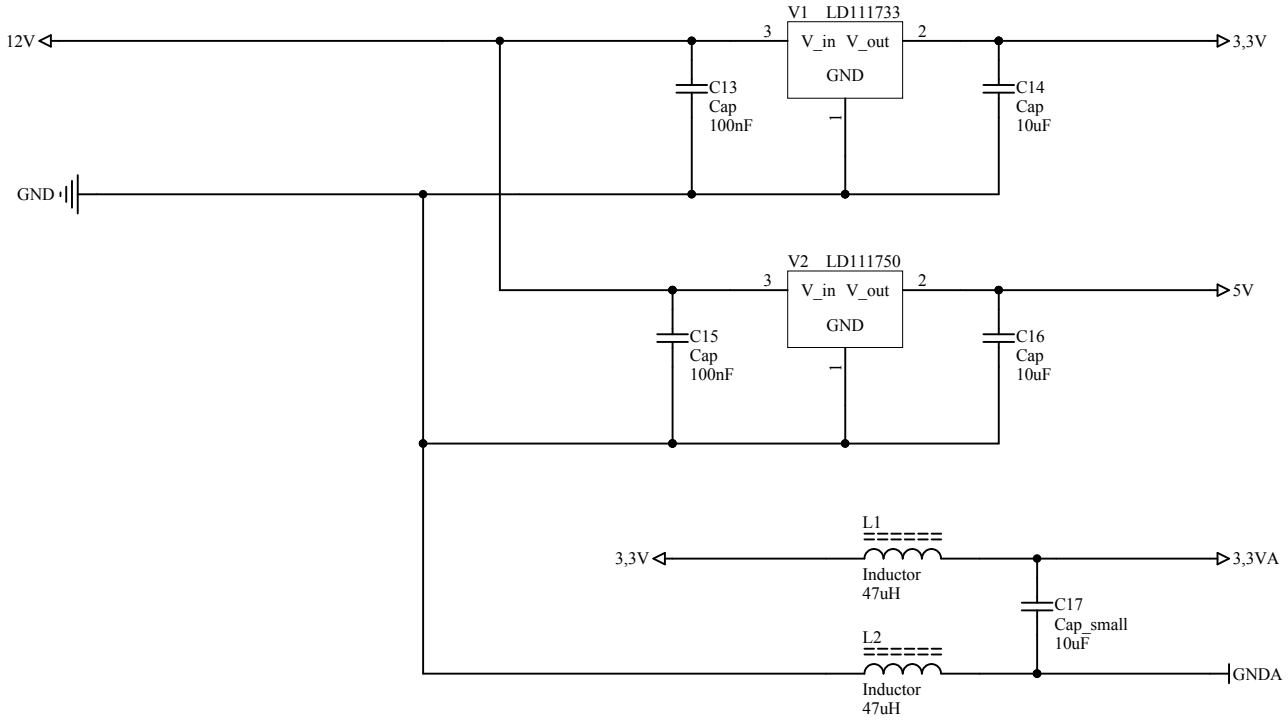
Title	**H-bridge M2**		
Size A4	Number **1.0**		Revision **1**
Date: 28.5.2014		Sheet of	**3/7**
File: D:\Dokumenty\..\menic_M2.SchDoc		Drawn By:	**Jiri Ctibor**



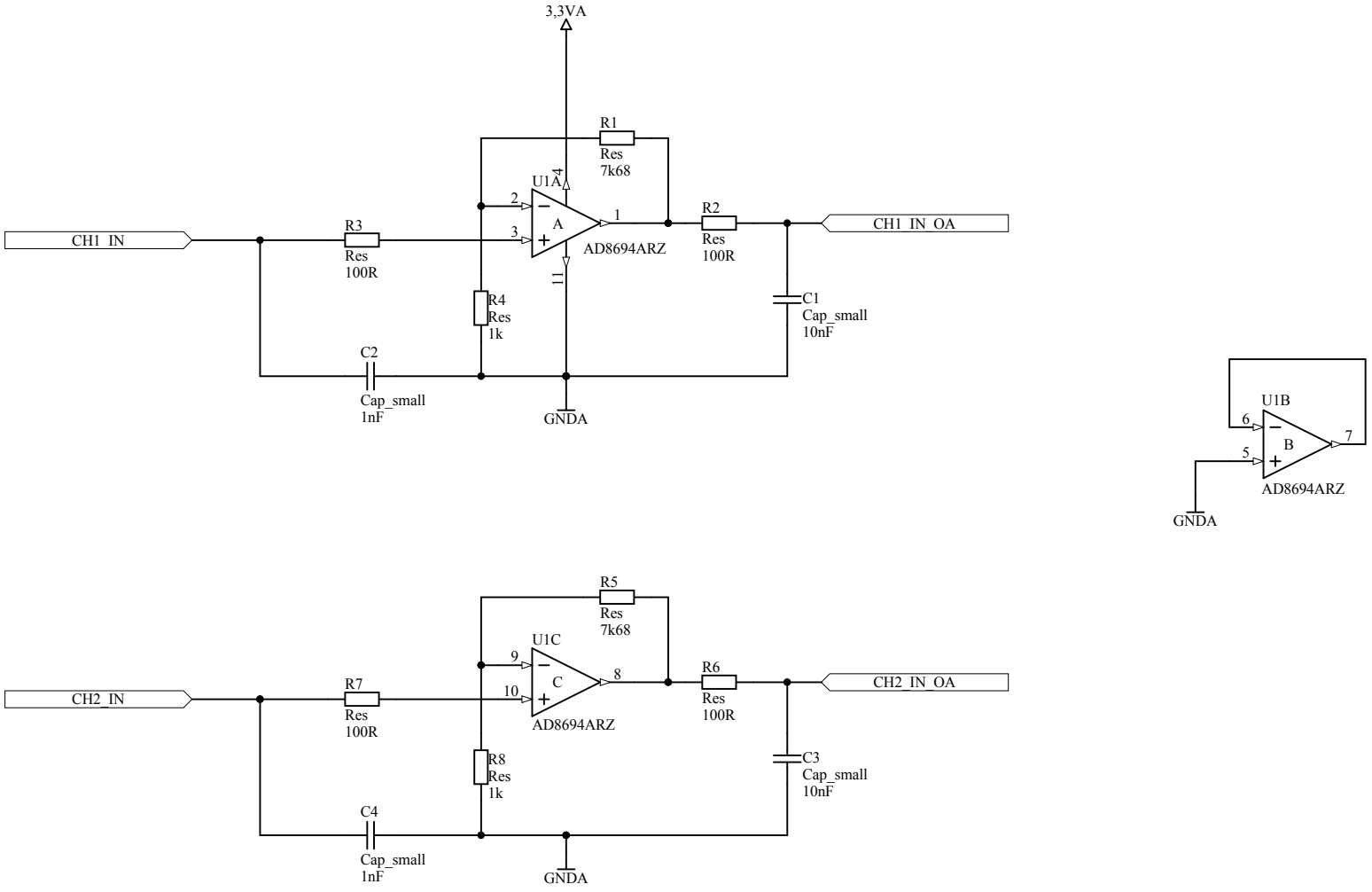
Title	**Battery measurment**		
Size A4	Number **1.0**	Revision **1**	
Date: 28.5.2014		Sheet of **7/7**	
File: D:\Dokumenty\..\mereni_napeti.SchDoc		Drawn By: **Jiri Ctibor**	



Title	**Processor circuit**		
Size A4	Number **1.0**		Revision **1**
Date: 28.5.2014		Sheet of	**1/7**
File: D:\Dokumenty\..\procesor_freescal.SchDoc		Drawn By:	**Jiri Ctibor**



Title	**Voltage reference**		
Size A4	Number **1.0**	Revision **1**	
Date: 28.5.2014		Sheet of **6/7**	
File: D:\Dokumenty\..\volt ref.SchDoc		Drawn By: **Jiri Ctibor**	



Title	**Analog inputs**		
Size A4	Number **1.0**		Revision **1**
Date: 28.5.2014		Sheet of	**5/7**
File: D:\Dokumenty\..\zesileni_analog.SchDoc		Drawn By:	**Jiri Ctibor**